An Exterior Communications Interface for the USNRC Consolidated Code

J. Mahaffy¹, C. Murray²

¹Penn State University, Applied Research Laboratory, P.O. Box 30, State College, PA 16804, USA
Tel: 814-863-4018, Fax: 814-865-8896, e-mail: jhm@psu.edu

²Penn State University, Applied Research Laboratory, P.O. Box 30, State College, PA 16804, USA
Tel: 814-865-2016, Fax: 814-865-8896, e-mail: cjm18@psu.edu

Summary

One of the requirements for the USNRC’s consolidated reactor safety analysis code is that it support parallel computations. One standard parallel calculation would speed calculations by distributing spatial regions (system components) to copies of the Consolidated Code running on separate processors. A more interesting form of parallel processing uses additional programs to provide special component or process models to extend the code’s capabilities. We have developed a code structure and communications protocol that supports both these forms of parallelism. The Consolidated Code uses a flexible request driven exterior communication interface. A set of negotiations at the beginning of a calculation determines the data to be transferred. Both the requesting and responding tasks record source and destination pointers for transfers in special transfer tables. Actual transfer of data is a very simple process, using the transfer tables to fill or empty interprocess buffers.

Introduction

Two design criteria for the USNRC consolidated reactor safety code have driven the use of parallel computing. The first is the standard desire for faster real-time turnaround of transient simulations. The resulting improvements in staff efficiency have always been important, and will continue to be significant for such applications as risk-informed regulation. The second is the need for a tool that can be easily extended as new modeling capabilities or user features are needed. The definition of a powerful Exterior Communications Interface (ECI) for the Consolidated Code permits addition of new capabilities through interaction with separate programs. A developer now need only understand the interface specifications and general features of the Consolidated Code to add capabilities. Detailed knowledge of internal coding is not necessary.

The Consolidated Code is designed to provide simulations of transients involving all aspects of a nuclear reactor’s primary and secondary coolant systems. However, the level of detail in its models may not be appropriate for all analyses. Greater detail might be required to capture some physical behavior, or a simpler component model might capture all relative phenomena at a lower computational cost. As examples, some simulations might need a detailed containment model, 3-D neutron kinetics, or an AP600 Core Makeup Tank (CMT) with resolution of surface thermal
stratification. None of these features are currently scheduled for the Consolidated Code, but the first two are currently available in other programs. In the case of the CMT, a relatively simple standalone program could be written to capture the thermal stratification. The Consolidated Code will permit tight coupling of its results with those from other special codes, through a well defined interprocess communication interface. In the discussion that follows, these separate computer programs will generally be referred to as "processes". This paper describes the context and defines the high level protocol for the process to process communication.

![Figure 1 Process Connectivity for a Reactor Analysis](image)

The basic model for operation of the interface is centered on the Consolidated Code. It is normally assigned central control of interprocess communications and key problem calculations. Satellite processes are spawned by the central one to provide supporting calculations. Figure 1 provides an illustration of this mode of operation. In this example the central process models the thermal-hydraulics of the reactor vessel. Two other Consolidated Code jobs model coolant loops in the reactor. This demonstrates that the Consolidated Code is not restricted to a central role, and can also be used as a satellite process. A fourth process provides a 3-D kinetics model, obtaining core fluid conditions and rod temperatures from the central process, and returning powers to the central process's rod conduction calculations. Another special process provides a detailed model of the containment, receiving mass and energy flows from the central process and one of the coolant loop calculations. The containment process returns fluid state information needed to compute the connecting flows. A key feature of the interface, illustrated with the containment, is that communication paths need not be routed through the central process. The ECI supports direct communication between satellite processes.

One limitation was recognized in design and installation of the ECI. The desires for increased speed and code extensibility are not compatible in all hardware and software environments. Speed penalties should be expected when parallel applications are ported to platforms which rely on normal network software and hardware for interprocess communications. Configurations such as
multiple PCs on a standard Ethernet, should only be regarded as development platforms, or used for calculations where run time is not as critical as developer or user time. Latency associated with message passing in such situations is relatively high. When the ECI is used to configure simulations with speed as a goal, the applications should be ported to a shared memory parallel (SMP) machine or distributed processors with a very high speed communications hardware, using appropriate supporting communications software.

**Computational Flow and Synchronization Points**

Before establishing communications links, the developer of a satellite application for the Consolidated Code needs to consider its general computational flow. Input processing establishes basic geometry, and state variables. An initialization phase maps the system configuration, and evaluates dependent variables not required in the basic state information (e.g. mean densities, specific internal energies, etc.). Next the code approximates the solution of the flow and conduction equations with a series of time steps. The following outline summarizes the key stages in a time step used by the Consolidated Code. Its structure largely reflects the use of the SETS [1],[2] numerical method. When the code is operating in semi-implicit mode stages associated with stabilizer equations are not present.

1. Setup
   a. Local Time Step Size
      Calculate time step size based upon restrictions placed by conditions in the regions covered by each process.
   b. Global Time Step Size
      All time step limits are collected by the central process and a final step size sent back to the other processes.
   c. Set Old Time Values
      Move new time results from the previous time step into the old time arrays. Generate basic old time only fluid properties such as viscosity, conductivity, and mean density.

2. Control System
   All signal variables, trips, and control blocks are evaluated. Signal variables may reference state variables in regions modeled by another process.

3. Correlation Evaluation
   Evaluate correlations for heat transfer and friction (wall and interfacial). This is separate from the basic time step setup because these correlations can and do involve basic fluid properties in adjacent components. Correlations may be component specific. This pass also includes special component models such as pump momentum source terms and valve area calculations.

4. Stabilizer Momentum Equation
   a. Evaluate Terms
      Coefficients and right hand sides for the momentum equations are evaluated for each face in the system. These linear equations have stabilizer velocities as the unknowns.
   b. Local Solution
      Solution for velocities within this process in terms of unknown velocities in regions evaluated by other processes.
   c. Central Solution
      Final solution of the reduced velocity equations either on the central process or on a designated special process.
d. Solution Storage
The global solution is stored in the local component data structures.

5. Semi-Implicit Equation Solution
   a. Velocity Dependency Equation
      Evaluate terms in the linear equations relating the new time velocities to the new time pressures.
   b. Evaluate average quantities at volume edges for use in computation of mass and energy fluxes.
   c. Evaluate Mass and Energy Equation Terms
      Mass and energy equations are linearized about current new time estimates. Coefficients and right hand sides for the equations are evaluated for each volume in the system
   d. Local Solution
      Solution for pressure changes are obtained in terms of unknown pressure changes in regions evaluated by other processes
   e. Central Solution
      Final solution of pressure equations either on the central process or on a designated special process
   f. Solution Storage
      New time temperatures, air partial pressure and void fraction are computed based on the global pressure solution. These are used to update all dependent fluid state information. All new variables are stored in the local component data structures.
   g. Local Convergence Checks
      Each participating process checks changes in primary variables or residuals against local convergence limits to determine if the non-linear equation solution is locally converged.
   h. Global Convergence Check
      The central process surveys the local convergence checks from all processes. If all have declared convergence, the calculation moves on to the stabilizer mass and energy equations. If not the calculation branches back to step 5b for re-linearization of the equations and another iteration of the Newton solution method. If the maximum permitted iteration count is exceeded, or water packing is detected, this sends a message to trigger the appropriate time step backup in all processes.

   a. Evaluate Terms
      Coefficients and right hand sides for the stabilizer mass and energy equations are evaluated for each volume in the system. These equations are linear in stabilizer macroscopic densities and macroscopic energy densities
   b. Local Solution
      Solution for variables in terms of unknowns in regions evaluated by other processes
   c. Central Solution
      Final solution of the reduced stabilizer equations either on the central process or on a designated special process
   d. Solution Storage
      The global solution is stored in the local component data structures, and new time void fractions computed. Checks are made for excessive void fraction change that may trigger a time step backup

7. Conduction Solution
   Given boundary conditions from fluid components, and power from tables or kinetics
calculations, the conduction equation is solved for metal temperatures in each heat structure in the system.

8. Central Status Check

The central process checks to see if an excessive void fraction change or heat transfer energy error requires a backup. If not, information is transmitted on whether or not it is time for various edits. Special processes must respond to a request for a restart dump, but other edits are optional.

Synchronization points for interprocess communications are not required at all of the above stages. Communication points are made available to support the most complex interaction, a SETS based solution of flow equations, spanning processes. In that case information must be passed at a number of points in a time step. Evaluation of time step is accomplished in a single special exchange. All satellite processes transmit the results of step 1a to the central process, which performs a serial calculation to select the system-wide time step limit. Beginning of time step values may be also exchanged at this first synchronization point. Control block values are transmitted after stage 2. At the end of stages 4b, 5d, and 6b, equations are passed to the central process for final solution. At the end of stages 4c, 5e, and 5c values of “exterior” variables are passed back to special processes to complete their equation solutions. After all stabilizer velocities have been obtained (stage 4d), they must be communicated between processes for use in the stage 5a of the semi-implicit step. At the same time any necessary information on heat fluxes from structures to fluid is transmitted across processes. Communication after setup of the velocity dependency equations is useful but not always required between processes using a semi-implicit method. These dependency relations are transmitted after step 5b along with the edge average quantities needed for mass and energy fluxes. Satellite processes communicate the state of convergence and any requests for time step backup to the central process after stage 5g. The central process in turn issues instructions to continue iterations, move on, or redo the time step after stage 5h. New time velocities at process junctions must also be passed at this point for further iteration or continuation to the stabilizer mass and energy equations. At the end of step 7 the central process receives information on whether or not a backup is required by other processes. After step 8 the central process may trigger a backup and request edits.

Two other transfer points are provided before the solution step. Some calculations and error checks performed by the Consolidated Code during initialization need information from other tasks. As a result an synchronization point named “Input” has been inserted before any component initialization subroutines are executed. Many quantities specifying system geometry need only be transmitted once in a calculation. These are may be transferred after all initialization has been completed, but before the first time step is started.

Simple satellite applications may only require data transfer at the beginning or end of each time step. Processes containing their own variation on the SETS numerical method will need to use all of the communication points listed above. Connection to a program using a fully implicit method to solve flow equations is not directly supported at this time. However, a fully implicit program may be modified without too much effort to behave like a semi-implicit or SETS method at a connection to another process. Templates of the above computational flow and communication schedule are provided with the Consolidated Code distribution to aid developers in scheduling interprocess communications.

The synchronization points listed above for communication also have a significance for deeper parallelism within the code. A developer can be assured that between any pair of these synchronization points, computations associated with a given mesh cell or cell face do not depend on
the results obtained for any other cell or face between the same pair of synchronization points. This permits usage of parallel compiler directives such as those in the OpenMP standard [4], to facilitate efficient use of the code on shared memory parallel processors. Loops exist within the code so that this parallelism may be implemented at the cell or component levels. Research is currently scheduled to determine the most effective way to take advantage of such parallel directives.

**Configuration of Data Transfers**

The ECI is based on dynamic configuration of data transfers. The content and timing of data transfers are established during the initialization, based upon exchange of requests between processes. Scheduling begins with transmission of the central process’s identification number to all processes. Each process reads its input files, and builds a list of system component numbers from which it needs information, but which are not included in its own input. Each then transmits the list of missing components to every other process. Each other process returns a subset of this list identifying the components for which it is responsible. Additional information is contained in this found component list providing information about the process’s treatment of the component. An abbreviation designates the nature of the component model (fluid, heat structure, neutronics, ...). This is used for consistency checks and error messages. When the component to component connection between processes involves fluid flow, two more pieces of information are transmitted. One indicates whether or not the component’s owner must control the solution of the full system of fluid equations. The other indicates whether the flow equations at the junction are to be explicit, semi-implicit, or SETS. A flow connection between processes also drives one more information exchange to establish the general form of the flow paths between components. Each satellite process sends a block of information to the central process uniquely specifying the exterior flow connections to its mesh, and indicates whether it must control the evaluation of momentum equations at these flow junctions. The central process returns a full map of flow connections, including a set of unique indices for cells and cell faces adjacent to the interprocess flow junctions. These indices are used in a fully coupled solution of the flow equations.

This initial negotiation imposes requirements on satellite processes, to recognize the concept of components as well as standard indexing conventions for a finite volume mesh. Most special purpose programs will only compute results for a region considered to be one component. However, for standardization and to permit multiple use of that program within a system simulation, the user must be able to assign arbitrary “component” numbers to label specific regions of the system. The user must also be able to assign unique numbers to identify flow junctions between processes.

Once initial information is provided to the central process, connections are checked for consistency (e.g. you can’t have a fluid flow connection to a heat conduction component). Component lists are checked to be sure that all have been located and a process identifier is directly associated with each component number. Where problems are detected, useful messages are printed for the code user. At this stage actual data transfer scheduling can begin. Based on known location of exterior components, each process transmits blocks of information to other processes requesting receipt of information. Each data request block contains variable information groups giving:

1) A pneumonic name for the variable requested (e.g. “alpha”, “p”, “t”);
2) The number of the component containing the information;
3) Cell or junction numbers locating the information;
4) The synchronization point at which the value should be transmitted (start of time step, end of each iteration, etc.)
Internally it notes the storage destination for each of these pieces of information, using a pointer assignment. Next it transmits a block of information to each connected process indicating information that it must transmit to that process’s data structure. This is currently needed for transfer of heat structure heat flux information to a fluid component.

Processes receive and store these requests. Now each possesses a complete list of information that it must transmit and receive at each synchronization point. These lists are stored in the form of transfer tables, implemented as Fortran 90 derived type arrays. Each element of the array contains a “from” and a “to” pointer variable (e.g. table(i)%to, table(i)%from). For transmissions the table’s “from” pointer is associated with the location in memory of a variable requested by another process, and the “to” pointer is associated with a location in an interprocess communication buffer that will be transmitted to the requesting process by a message passing service. This pointer association is a form of dynamic equivalencing. Any future reference to table(i)%from will directly access the memory location for the information scheduled by the corresponding exterior data request. At this stage of the scheduling all processes understand the exact sequence of information that they must transmit to other processes, and understand the meaning of every string of information received from other processes. Time steps can begin.

Data Transfers

Internal details of data transfer can vary from program to program. Key features of the data transfer between two Consolidated Code processes are summarized in Figure 2. The first stage is internal to one process. It is table driven and very similar to the table driven data transfer used internally to communicate information between components. The export transfer table is built during the initial scheduling described above. At the actual time of a scheduled transfer, the export transfer buffer is filled with a loop structure of the form:

```
DO i = 1, SIZE(table)
   table(i)%to = table(i)%from
ENDDO
```

(Step 1 in Figure 2). The use of pointers in the above loop amounts to a form of indirect indexing, which operates very quickly on modern CPU’s.

The actual process to process communication (Step 2 in Figure 2) is facilitated by the transfer buffers. In the current version of the code, contents of the buffers are transferred between processes with calls to PVM [3]. The implementation of this transfer is restricted to a single module containing low level subroutines, and can be easily adapted to other libraries (e.g. MPI-2, [5]) supporting interprocess communication. Currently, only 15 PVM subprograms are utilized in a total of 30 PVM calls.

The final stage of data transfer is a mirror image of the first stage. Here the initial scheduling has assigned pointers in an “import” derived type transfer array. The “from” pointers are associated with elements of import transfer buffer, and the “to” pointers associated with the final storage locations for the data. Step 3 of Figure 2 consists of execution of the above DO loop, using an import transfer table, to place data in its final destination.
Figure 2 Table driven data transfer from a fluid component to a heat structure.

Figure 2 does not capture the fact that communication is also running in the other direction at the same time (reverse the headings “Process 1” and “Process 2”). If both processes exist on a shared memory parallel computer, the possibility exists for the export transfer buffer of one to be the same block of memory as the import buffer of the other process. This variation can be implemented under the MPI-2 standard, and has a potential for significant time savings for applications with unusually large data transmission requirements. Unfortunately, MPI-2 has not yet been widely implemented on parallel computers. Implementation of a shared buffer now would generally require use of machine specific coding.

Solution Procedure for Multiprocess Flow Modeling

When more than one task is responsible for fluid flow modeling, some form of implicit coupling should be maintained between the tasks to avoid numerical instabilities. If proper data naming conventions are followed, the Consolidated Code will correctly couple the solution of semi-implicit or
SETS numerical methods across all processes contributing to the fluid flow. Provisions also exist within the communications interface for another process to take control of the fluid equation solution. This represents a much more difficult programming effort for the application developer, but might be necessary for a special process which provides detailed CFD modeling of a specific region within the full system.

Solution of the flow equations is based upon an agreement between tasks on the location of “exterior variables” to be used in a two stage reduction of the equations. Figure 3 illustrates special variable assignments performed during a multiprocess solution of flow equations. The upper half of the loop is modeled with one process, the lower half is simulated with a flow model in a parallel process. To shorten notation in the following discussion, cells in the figure have been given an absolute numbering convention. In terms of component numbering, cells 1-4 in this figure can be considered cells 1-4 of PIPE 1 (modeled by the first process), and cells 5-8 in the figure are cells 1-4 of PIPE 2 (modeled on the second process). For the multiprocess solution, one variable is assigned on each side of each junction between tasks. For the semi-implicit pressure equation the exterior variables \( dp_{Ext\, 1} \), \( dp_{Ext\, 2} \), \( dp_{Ext\, 3} \), and \( dp_{Ext\, 4} \) correspond directly to the solution variables \( dp \) at volumes with absolute indices 1, 4, 8, and 5 respectively.

Selection of exterior variables for cell edges associated with the SETS stabilizer momentum equations is somewhat more complex. Again two exterior variables are associated with each interprocess flow junction. At a given junction, the task responsible for evaluation of the junction momentum equations associates its exterior variable with that junction’s velocity. The task not responsible for the interprocess junction momentum equations evaluates its exterior variable at the next face in from the junction face. For the example in Figure 6, the exterior velocity variables \( v_{Ext\, 1} \), \( v_{Ext\, 2} \), \( v_{Ext\, 3} \), and \( v_{Ext\, 4} \) correspond directly to the stabilizer velocities at edges with absolute indices 1, 5, 8, and 6 respectively.

The first stage of the solution is local to each process, and follows the normal equation solution procedure with the addition of terms on the right hand side of each equation to account for the unknown effects of variables in adjacent processes. For the first process in the Figure 3 example, the system of pressure equations would have the form:

\[
A \ dp = b - a_{Ext\, 3} \ dp_{Ext\, 3} - a_{Ext\, 4} \ dp_{Ext\, 4}.
\]
which is solved to give:
\[ dp_i = dp_{o,i} - a_{i,3} dp_{Ext 3} - a_{i,4} dp_{Ext 4}. \]
A similar set of solutions are generated by the second process in the form:
\[ dp_i = dp_{o,i} - a_{i,1} dp_{Ext 1} - a_{i,2} dp_{Ext 2}. \]

For the second stage of the solution, the reduced equations adjacent to each inter-process connection are collected by the solver process:
\[ dp_1 = dp_{o,1} - a_{1,3} dp_{Ext 3} - a_{1,4} dp_{Ext 4} \]
\[ dp_2 = dp_{o,1} - a_{1,3} dp_{Ext 3} - a_{1,4} dp_{Ext 4} \]
\[ dp_3 = dp_{o,1} - a_{1,3} dp_{Ext 3} - a_{1,4} dp_{Ext 4} \]
\[ dp_4 = dp_{o,1} - a_{1,3} dp_{Ext 3} - a_{1,4} dp_{Ext 4} \]

Recognizing the identity relationships between local variables and exterior variables, and gathering unknowns on the left side, these equations become:
\[ dp_{Ext 1} + a_{1,3} dp_{Ext 3} + a_{1,4} dp_{Ext 4} = dp_{o,Ext 1} \]
\[ dp_{Ext 2} + a_{1,3} dp_{Ext 3} + a_{1,4} dp_{Ext 4} = dp_{o,Ext 2} \]
\[ dp_{Ext 3} + a_{1,3} dp_{Ext 1} + a_{1,4} dp_{Ext 4} = dp_{o,Ext 3} \]
\[ dp_{Ext 4} + a_{1,3} dp_{Ext 1} + a_{1,4} dp_{Ext 4} = dp_{o,Ext 4}. \]

This closed set of equations is solved, and the final values of the exterior variables are transmitted back to the separate tasks where a final substitution step generates values for all local pressure variations. A similar series of algebraic operations is used for all equations associated with SETS stabilizer steps. Schemes can be devised that cut the number of “exterior variables”. This one was selected for relative simplicity, and associated ease of coding necessary data transfers.

By default the Consolidated Code takes responsibility for the above solution procedure. If only Consolidated Code processes contribute to the simulation, the whole solution is transparent to the developer, or user. If a developer writes an application to participate in a flow solution, he or she needs only pay attention to the naming conventions for terms in the above equations, to let the central process automatically generate the solution.

## Constructing Applications

The ECI consists of eleven Fortran 90 modules structured to minimize changes required for use with new applications. Only one of these modules contains subroutines directly called by the developer’s application. One of these subroutines provides the initial connection with other processes, and four others support the scheduling of data transfers. None of these five subroutines requires arguments. Actual data requests are made by calling a sixth subroutine with arguments specifying the name and location (component, and mesh indices) of the missing information, and providing a pointer to the destination of the information once it is transferred.

One of the ECI modules contains all application specific coding, which provides information on the data structure and problem configuration. One subroutine in this module must be rewritten if the application responds to information requests from other programs. This is the subroutine that interprets the variable name and location indices, and assigns a pointer to the correct location of information in the local data structure. Another subroutine must be modified to respond to inquiries from other processes about components that they are missing. If the application contains component models used by other
processes a short subroutine must be modified to provide basic information about the component. If the application contains a flow model, a simple subroutine must be modified to provide information about flow junctions, and another one altered to use results of the multiprocess flow solution. For many applications, code changes are relatively minimal and can often be adapted quickly from examples in the Consolidated Code distribution package. For an application with a very detailed flow model and unique data structure for fluid state variables, changes could run up to 1000 lines of Fortran.

The ECI has a large number of potential applications beyond those suggested by Figure 1. One class of application provides more detailed modeling than is available in the Consolidated Code. The ECI could for example be used to couple a detailed CFD model of a limited region in a reactor, with a standard simulation of flow in the balance of the system. Some caution would have to be exercised in this instance to provide a good velocity profile to the inlet of the CFD grid. This might involve actual CFD modeling of a larger region than first expected, and/or application of assumed fully developed velocity profiles to the interface between 1-D modeling from the Consolidated Code and 3-D modeling by the CFD code.

A second application class consists of simplified component or process models. These are designed to save time and frequently increase accuracy by using a simpler, but more directed model than is available within the consolidated code. As an example of this capability, we have extracted the accumulator model from RELAP5 [6] into a separate application. Its direct modeling of gas expansion and motion of the liquid surface is simpler and more accurate than use of the Consolidated Code’s full two-fluid model. This is provided with the code distribution as an example of exterior linkage to other flow models.

Enhanced user interfaces are another class of application that can utilize the ECI. The code distribution provides an application that allows users to provide their own control systems via a Fortran subroutine, and another application that recovers values of any requested set of variables for special post-processing. Work is also planned to permit direct user interaction during a calculation via the graphical user interface, Symbolic Nuclear Analysis Package, currently under development. This plant analyzer type capability will display selected variables as the calculation evolves, and allow user intervention to alter the course of the transient by changing valve settings, triggering a SCRAM, or tripping a pump.

Conclusions

The Exterior Communications Interface (ECI) provides a powerful and relatively convenient method to extend the capabilities of the USNRC’s Consolidated Code, without internal modifications. Application developers need to learn the general computational flow of the code, and obtain a dictionary of variable names to make and respond to information requests. Since they do not need to develop expertise on the code’s internal structure, they can make productive contributions in a relatively short period of time.

Users and developers need to remember that this feature does not automatically provide major run time improvements for system simulations. Direct application of the interface to faster calculations first requires that a user split the system input into input for multiple linked subsystems with balanced computational loads. It also requires the availability of suitable parallel hardware, preferably a shared memory parallel processor. In the future, the work associated with the ECI will provide more automated methods to efficiently use parallel computers. Restructuring of computational flow associated with the
installation of the ECI has provided many opportunities for use of directives such as those in the OpenMP standard to enable automatic distribution of the computational load across many processors.

References


